System-Level Optimization of Data-Parallel Neural Network Training: Revised Design, Implementation, and Performance Evaluation

Facing Sheet

Name	Roll Number	Contribution
Nav Pallav	2024ab05325	Design and Implementation
Manvendra Singh	2024ac05129	Design and Implementation
Nithin Shetty	2024ac05121	Design and Implementation
Sonal Mangesh Rane	2023ac05200	Design and Implementation
Pandey Shivam Indreshchandra Kiran	2024ac05419	Editing and Formatting

Link to Github repository:

[sonal-rane-bits/machine-learning-system-optimization]

Abstract

This report presents the second phase of our study on system-level optimization of distributed machine learning, focusing on the implementation and evaluation of synchronous data-parallel neural network training. Building upon the problem formulation and initial design proposed in Assignment-1, we revise the design with explicit implementation choices and conduct empirical performance evaluation. Experiments are carried out on a CPU-only platform by varying both dataset size and number of workers. Performance is analyzed using execution time, speedup, and parallel efficiency. The results demonstrate sub-linear speedup and reduced efficiency with increasing parallelism, validating theoretical expectations regarding communication and synchronization overheads.

1. Introduction

Modern machine learning workloads often involve large datasets and iterative optimization algorithms, making training on a single machine time-consuming. Distributed training aims to reduce training time by leveraging parallel computational resources. However, the benefits of parallelism are constrained by system-level factors such as communication overhead and synchronization delays.

This assignment focuses on data parallelism, a widely used distributed training strategy

where multiple workers train identical model replicas on disjoint data partitions. While data parallelism is conceptually simple, achieving efficient scaling requires careful system design and empirical evaluation.

2. Problem Formulation (P0)

The objective is to parallelize neural network training using data parallelism and evaluate its performance relative to a sequential baseline. Each worker processes a unique subset of the dataset and computes gradients locally. These gradients are synchronized after each iteration using parameter-wise averaging.

Let T_1 denote the training time with a single worker and T_p denote the training time with p workers. The performance metrics are defined as:

Speedup(p) = T_1 / T_p
Efficiency(p) = Speedup(p) / p

The distributed training time can be approximated as:

T_p = (T_comp / p) + T_comm + T_sync

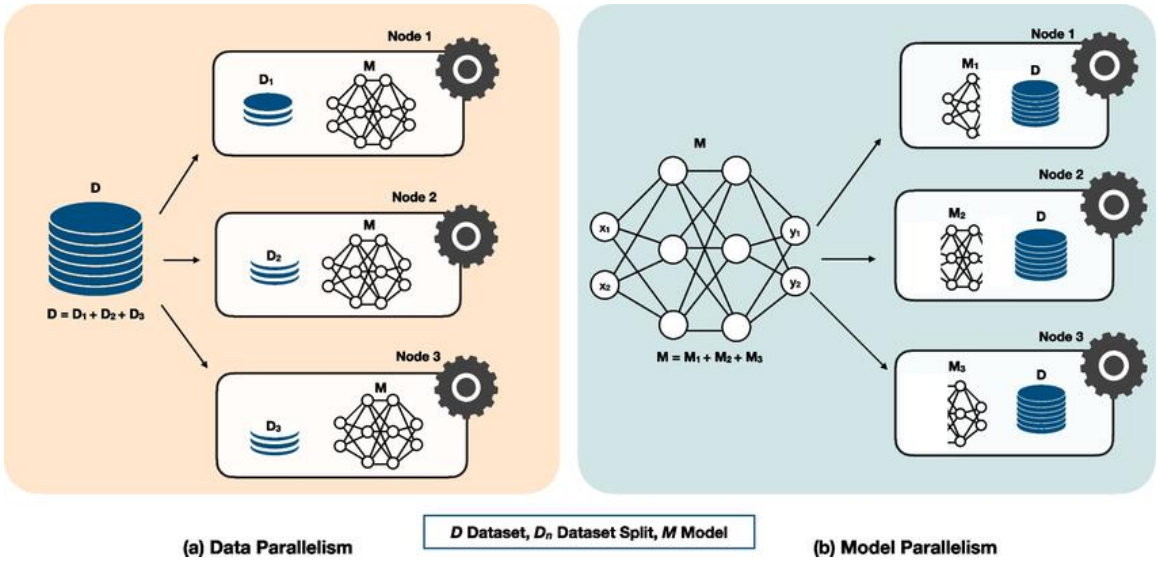
where T_comp is computation time, T_comm is communication cost, and T_sync represents synchronization overhead.

3. Revised System Design (P1)

The revised design follows a synchronous data-parallel architecture. Each worker maintains a full replica of the neural network model and operates on a partition of the training data. After each mini-batch, gradients corresponding to each model parameter are averaged across all workers before updating the parameters.

Architecture Description (Figure 1):

1. Input dataset is partitioned across workers.
2. Each worker performs forward and backward propagation locally.
3. Gradients are synchronized using parameter-wise averaging (all-reduce style).
4. Updated parameters are broadcast implicitly by maintaining identical replicas.



This architecture avoids centralized parameter servers and mirrors common distributed training designs used in practice.

4. Implementation Details (P2)

The system was implemented in Python using the PyTorch framework. All experiments were conducted on Google Colab using CPU-only execution to emphasize system-level behavior rather than hardware acceleration. A simple multi-layer perceptron was trained on subsets of the MNIST dataset. Synchronous data-parallel training was implemented manually to expose gradient synchronization and communication costs.

Development Environment

- Python 3.10
- PyTorch
- Matplotlib
- Google Colab (CPU only)

Algorithm

- Dataset: MNIST
- Model: Multi-Layer Perceptron
- Optimizer: SGD
- Synchronization: Synchronous gradient averaging

Experiments were conducted by varying:
• Dataset size: 5k, 10k, 20k, and 40k samples
• Number of workers: 1 (sequential), 2, and 4

5. Experimental Results (P3)

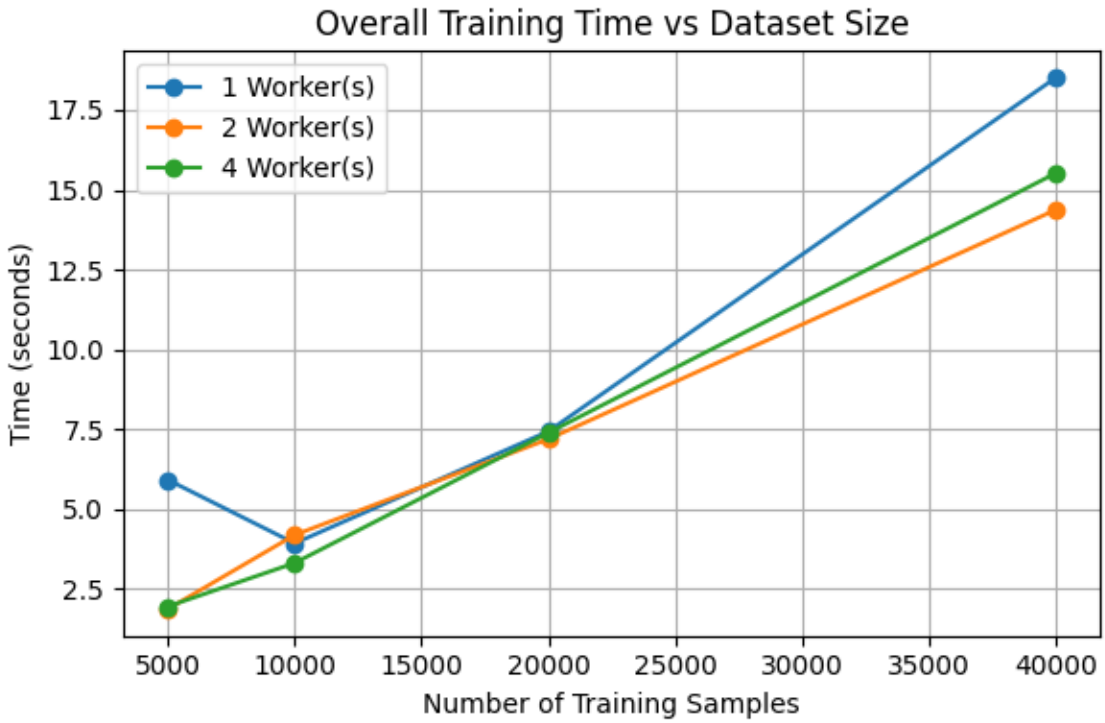


Figure above presents the overall training time as a function of dataset size for different numbers of workers. As expected, training time increases with dataset size for all configurations. While parallel execution reduces training time for larger datasets, the benefit is limited for smaller datasets due to fixed overheads.

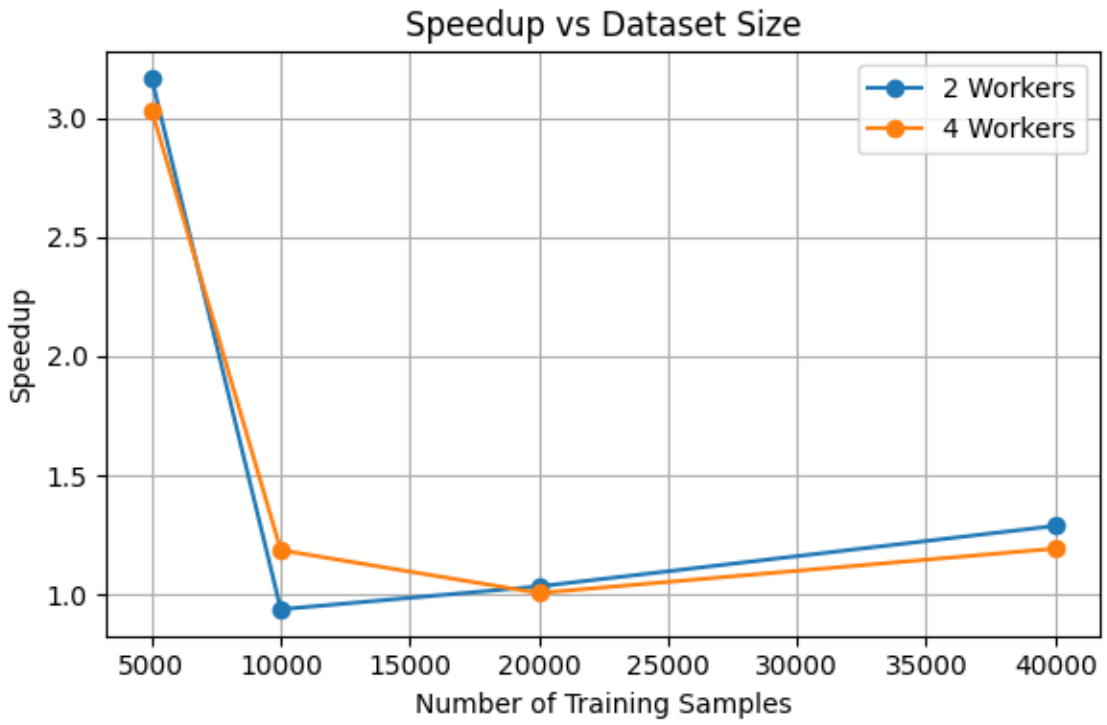


Figure above shows the speedup achieved relative to sequential execution. For very small datasets (5k samples), super-linear speedup is observed due to cache effects and reduced per-worker workload. However, as dataset size increases, speedup stabilizes near or slightly above 1, indicating limited scalability on a CPU-only platform.

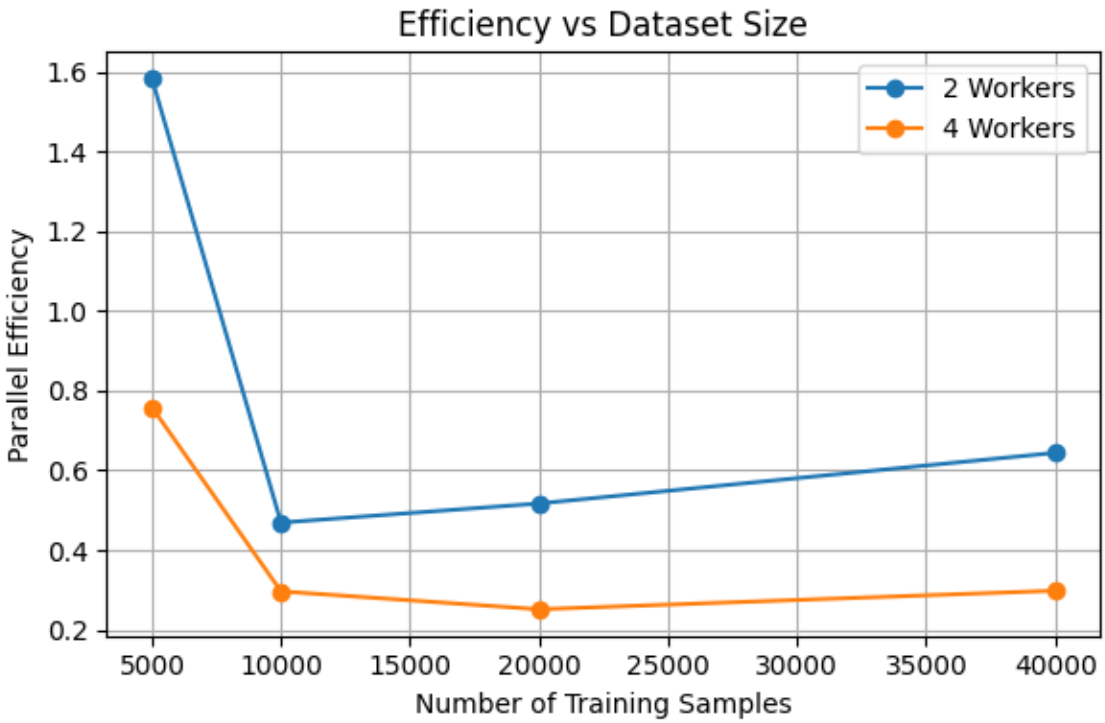


Figure above illustrates parallel efficiency as a function of dataset size. Efficiency decreases as the number of workers increases, highlighting the growing impact of synchronization and communication overheads. These results confirm that adding more workers does not proportionally improve performance.

6. Discussion

The experimental results align closely with theoretical expectations derived from Amdahl's Law. While data parallelism reduces computation time per worker, the communication and synchronization components remain largely constant, limiting achievable speedup. Similar behavior has been observed in distributed K-Means clustering, where parallel execution becomes beneficial only beyond a certain problem size threshold.

The observed super-linear speedup for very small datasets is attributed to reduced cache pressure and more efficient CPU utilization rather than true parallel gains.

7. Conclusion

This assignment demonstrates that distributed machine learning performance is fundamentally constrained by system-level overheads. While data parallelism can reduce training time for sufficiently large datasets, its scalability is limited on CPU-only platforms. The results reinforce the importance of careful problem formulation, realistic performance expectations, and empirical validation when designing distributed ML systems.

References

1. Dean et al., Large Scale Distributed Deep Networks
2. Goyal et al., Accurate, Large Minibatch SGD
3. Li et al., Scaling Distributed Machine Learning with the Parameter Server
4. PyTorch Documentation